# Programowo-Sprzętowa Realizacja Algorytmów: Wyznaczanie i śledzenie punktów charakterystycznych

Wojciech Gumuła, Rafał Prusak

6 kwietnia 2016

# Spis treści

# Rozdział 1

# Metody wykrywania punktów charakterystycznych.

Ważnym zagadnieniem w systemach wizyjnych jest pozyskiwanie informacji z obrazu, które mogą być wykorzystane w celu inicjacji dalszych działań aplikacji.

Wyznaczanie punktów charakterystycznych ma na celu wybór „interesujących”<<<<<<< HEAD elementów bądź rejonów obrazu oraz uzysanie charakterystyki specyficznej dla danej aplikacji. Jest to często niskopoziomowa składowa bardziej złożonych algorytmów. Uzyskanie punktów charakterystycznych pozwala na rozpoznawanie kształtów, wyszukiwanie elementów pasujących do wzorca czy też śledzenie obiektów w obrazie wideo.======= elementów bądź rejonów obrazu oraz uzyskanie charakterystyki specyficznej dla danej aplikacji. Jest to często niskopoziomowy element bardziej złożonych algorytmów. Uzyskanie informacji na temat punktów charakterystycznych pozwala na rozpoznawanie kształtów, wyszukiwanie elementów pasujących do wzorca czy też śledzenie obiektów w obrazie wideo.>>>>>>> 7ad2c858d63c3aa7d03ac25900b75c57c9eab0ca

Istnieje kilka rodzajów punktów uznawanych jako charakterystycznych:

- krawędzie,
- narożniki,
- <<<<<<< HEADkrople(ang „blobs”).=======plamy.>>>>>>> 7ad2c858d63c3aa7d03ac25900b75

Szczególną uwagę poświęcono zagadnieniu wykrywania narożników. Punkty te powstają w miejscu krzyżowania się lub gwałtownej zmiany kształtu krawędzi. Z tego powodu, w punkcie narożnym dochodzi do nagłej zmiany wartości gradientu obrazu, co znacząco ułatwia jego poszukiwanie.

## 1.1 Metoda Harrisa

Algorytm Harrisa polega na analizie jasności obrazu i wyszukiwaniu zmian gradientu.

Poszukiwane są odchylenia jasności:

$E(u,v) = \sum_{x,y} w(x,y)[I(x+u, y+v) - I(x,y)]^2$

gdzie:

- $w(x,y)$ - okno w punkcie $(x,y)$ o wymiarach $u \times v$,
- $I(x,y)$ - jasność w punkcie$(x,y)$,
- $I(x+u, y+v)$ - jasność w drugim punkcie okna.

Poszukujemy znaczącego odchylenia, więc celem jest maksymalizacja:

$\sum_{x,y}[I(x+u, y+v) - I(x,y)]^2$

Po rozwinięciu w szereg Taylora:

$E(u,v) \approx \sum_{x,y}[I(x,y) + uI_x + vI_y - I(x,y)]^2$

Dokonujemy skrócenia:

$E(u,v) \approx \sum_{x,y} u^2 I_x^2 + 2uvI_xI_y + v^2 I_y^2$

Równanie można zapisać w postaci macierzowej:

$E(u,v) \approx [u\, v] M [\begin{smallmatrix} u \\ v \end{smallmatrix}]$

z macierzą postaci:

$M = \begin{matrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{matrix}$

Dla każdego okna wyliczana jest wartość wyrażenia:

$R = det(M) - k(trace(M))^2$

gdzie:

- $det(M) = \lambda_1\lambda_2$
- $trace(M) = \lambda_1 + \lambda_2$

Wartość wyrażenia $R$ powyżej pewnej wartości granicznej oznacza występowanie narożnika.

### 1.1.1 Implementacja w OpenCV

Poniższy kod prezentuje wykrywanie narożników za pomocą biblioteki OpenCV:

```
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include <iostream>
#include <stdio.h>
#include <stdlib.h>

using namespace cv;
using namespace std;

Mat src, src_gray;
int thresh = 200;
int max_thresh = 255;

char* source_window = "Source image";
char* corners_window = "Corners detected";

void cornerHarris_demo( int, void* ) {
    Mat dst, dst_norm, dst_norm_scaled;
    dst = Mat::zeros(src.size(), CV_32FC1);

    int blockSize = 2;
    int apertureSize = 3;
    double k = 0.04;

    cornerHarris(src_gray, dst, blockSize, apertureSize, k, BORDER_DEFAULT);

    normalize(dst, dst_norm, 0, 255, NORM_MINMAX, CV_32FC1, Mat());
    convertScaleAbs( dst_norm, dst_norm_scaled );

    for (int j=0; j<dst_norm.rows ; j++) {
        for (int i=0; i<dst_norm.cols; i++) {
            if((int)dst_norm.at<float>(j,i) > thresh)
                circle(dst_norm_scaled, Point(i, j), 5, Scalar(0), 2, 8, 0);
        }
    }

    namedWindow(corners_window, WINDOW_AUTOSIZE);
    imshow(corners_window, dst_norm_scaled);
}

int main( int argc, char** argv ) {
    src = imread(argv[1], 1);
    cvtColor(src, src_gray, COLOR_BGR2GRAY);

    namedWindow(source_window, WINDOW_AUTOSIZE);
    createTrackbar("Threshold: ", source_window, &thresh, max_thresh, cornerHarris_demo);
    imshow(source_window, src);
    cornerHarris_demo(0, 0);
    waitKey(0);
    return(0);
}
```

Rysunek 1.1: Wykrywanie narożników metodą Harisa w OpenCV

## 1.2 Metoda Susan

Algorytm SUSAN (ang. Smallest Univalue Segment Assimilating Nucleus) opiera się na znajdowaniu pikseli, które mają w swoim otoczeniu małą liczbę punktów o zbliżonej jasności.

$$c(r, r_0) = \begin{cases} 1 & gdy\, |I(r) - I(r_0)| \leq t \\ 0 & gdy\, |I(r) - I(r_0)| > t \end{cases}$$

gdzie:

- $I$ - jasność obrazu,
- $r_0$ - piksel centralny,
- $t$ - próg podobieństwa pikseli

Liczbę pikseli podobnych wyznaczamy następująco:

$$D(r_0) = \sum_{r \epsilon N(r_0)} c(r, r_0)$$

gdzie $N(r_0)$ to otoczenie piksela centralnego.

Aby punkt był uznany za narożnik, co najwyżej połowa punktów jego otoczeniu może być do niego podobna.

## 1.3 Metoda Fast

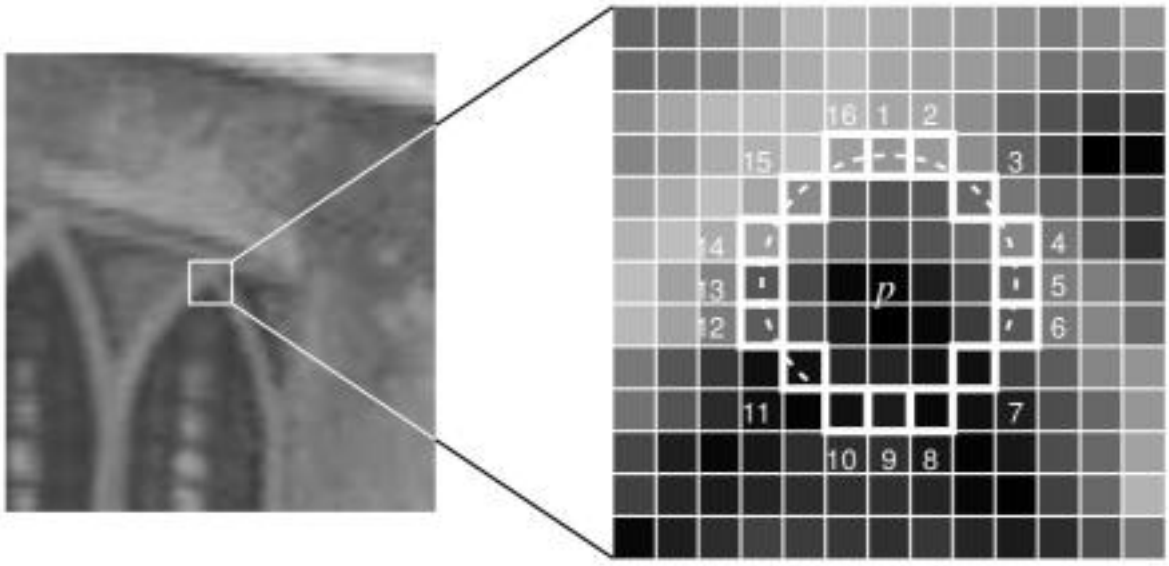
Algorytm Fast (ang. Features from Accelerated Segment Test) opiera się, podobnie jak Susan, na przeszukaniu otoczenia piksela.

Opis algorytmu:

1. Wybieramy piksel $p$ o poziomie jasności $I_p$.
2. Wybieramy wartość progu $t$.
3. Wybieramy okrąg o promieniu $r = 16$ pikseli.
4. Piksel $p$ jest narożnikiem jeżeli na okręgu znajduje się ciąg$n$ pikseli jaśniejszych niż $I_p + t$ lub ciemniejszych niż $I_p - t$.

Algorytm ten został udoskonalony przez dodanie testu, który pozwala bardzo szybko odrzucić dużą liczbę punktów nie będących narożnikami. W tym teście sprawdzana jest jasność pikseli 1, 9, 5, 13.

Dobór okręgu został przedstawiony na rysunku 1.2.




Rysunek 1.2: Wybór otoczenia piksela w metodzie FAST.

### 1.3.1 Implementacja w OpenCV.

Poniższy kod prezentuje wykrywanie narożników metodą Fast za pomocą biblioteki OpenCV:

```
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/features2d.hpp>
#include <vector>
#include <iostream>

using namespace cv;
using namespace std;

Mat src, src_gray;
int thresh = 20;
int max_thresh = 50;

char* source_window = "Source image";
char* corners_window = "Corners detected";

void Fast_demo(int, void *) {
    std::vector<KeyPoint> points;
    FAST(src_gray, points, thresh);

    auto temp = src_gray.clone();

    for(auto keyPoint: points) {
        int x = keyPoint.pt.x;
        int y = keyPoint.pt.y;
        Point p{x, y};
        circle(temp, p, 5, Scalar(0));
    }

    namedWindow(corners_window, WINDOW_AUTOSIZE);
    imshow(corners_window, temp);
}

int main(int argc, char** argv) {
    src = imread(argv[1], 1);
    cvtColor(src, src_gray, COLOR_BGR2GRAY);
    namedWindow(source_window, WINDOW_AUTOSIZE);
    createTrackbar("Threshold: ", source_window, &thresh, max_thresh, Fast_demo);
    imshow(source_window, src);
    Fast_demo(0, 0);
    waitKey(0);
    return(0);
}
```

Rysunek 1.3: Wykrywanie punktów charakterystycznych metodą Fast(t=50).

# Rozdział 2

# Analiza istniejących rozwiązań

## 2.1 Dopasowywanie punktów w obrazie przestrzennym oparte na wykrywaniu narożników.

Na podstawie artykułu: „A New Stereo Matching Method Based on Sub-pixel Corner Detection"(Dou Zhao, Ding Liu, Yanxi Yang).

Znajdowanie i dopasowywanie punktów charakterystycznych odgrywa ważną rolę w przetwarzaniu obrazu 3D, wykrywaniu i śledzeniu obiektów. W tych zagadnieniach warunkiem koniecznym jest wysoka wydajność systemu oraz dokładność detekcji.

W zagadnieniach praktycznych wyróżnia się 3 podejścia: dopasowywanie obszarów (ang. „Area-based"), dopasowywanie punktów charakterystycznych oraz dopasowywanie fazowe (ang. „Phase-based"). Pierwsza metoda zapewnia wysoką precyzję, ale posiada znaczące wady w postaci trudności wyliczania rozmiaru okna oraz znacznego nakładu obliczeń. Kolejna metoda wyróżnia się odpornością na na zakłócenia i wysoką wydajnością. Niestety, cechuje się ona również zmniejszoną precyzją. Ostatnia metoda jest w stanie usuwać szumy wysokiej częstotliwości oraz jest łatwa do implementacji za pomocą obliczeń równoległych.

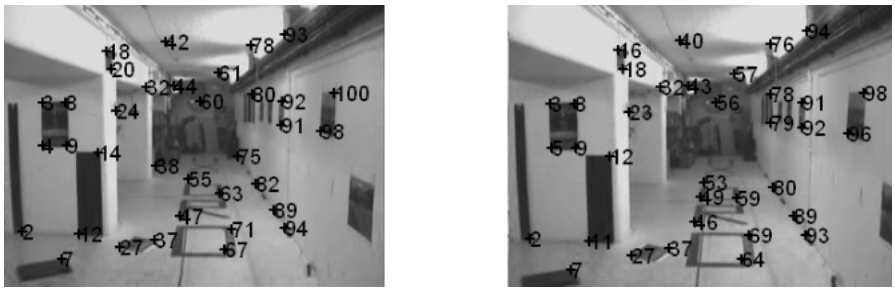
W celu poprawy dopasowywania metody punktów charakterystycznych opracowana została podwójna metoda Harisa z dopasowywaniem punktów opartym na poziomie jasności. Algorytm ten składa się z następujących kroków:

1. Konwersja obrazu na skalę szarości i filtracja.
2. Wykrycie narożników standardową metodą Harrisa.
3. Podział obrazu na mniejsze bloki rozmiaru $n \times n$ zawierające wykryte krawędzie.

4. Interpolacja jasności każdego z regionów.

Dopasowywanie punktów odbywa się za pomocą wyliczania charakterystyki $R$ każdego punktu i analizie stosunku wartości charakterystyk. Stosunek zbliżony do 1 oznacza dokładne dopasowanie.

Poniższy obraz prezentuje rezultat algorytmu:



Rysunek 2.1: Dopasowanie punktów charakterystycznych.

W zaprezentowanym przykładzie błąd wyniósł około 38%.

## 2.2 Odporny algorytm dopasowywania szablonów za pomocą wykrywania narożników dla systemów wizyjnych robotów.

Na podstawie artykułu „Robust Template Based Corner Detection Algorithms for Robotic Vision".

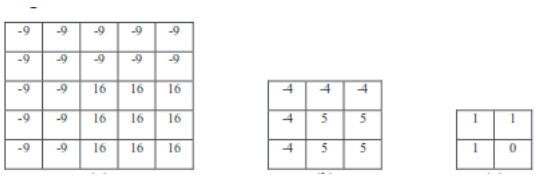
Wykrywanie narożników z powodu wydajności i niskiej złożoności obliczeniowej znalazło szereg zastosowań w systemach wizyjnych robotów. W połączeniu z odpowiednim zaprojektowaniu szablonów poszukiwanych elementów można uzyskać bardzo efektywne aplikacje. W artykule zarezentowano dwa podejścia do tematu: detekcja na podstawie dopasowywania narożników(MBCD - ang. „matching based corner detection") oraz dopasowywanie na podstawie korelacji punktów(CBCD - ang. „correlation based corner detection").

Z powodu ograniczonych zasobów sprzętowych w systemach wizyjnych robotów kluczowym elementem jest dobór algorytmu i poszukiwanych obiektów. Systemy te muszą działać na podobnej zasadzie co układ wzrokowy człowieka: być szczególnie czułym na elementy o wysokiej częstotliwości, takie jak krawędzie oraz narożniki.

Istniejące metody wykrywania punktów charakterystycznych można podzielić na dwie grupy: oparte na obliczaniu gradientów lub poszukiwaniu podobieństw. Pierwsza grupa algorytmów jest czuła na zakłócenia. Druga grupa algorytmów opiera się na poszukiwaniu podobieństw pomiędzy pikselami.

W wykorzystaniu szablonów zasadnicze znaczenie mają dwie kwestie: jakiego typu użyć szablon oraz jak zmierzyć różnice. Przykładowe szablony do

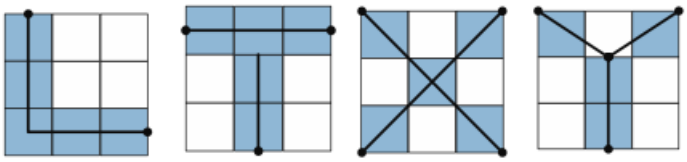
wykrywania rogów prezentuje poniższy obraz:




Rysunek 2.2: Szablony do wykrywania narożników.

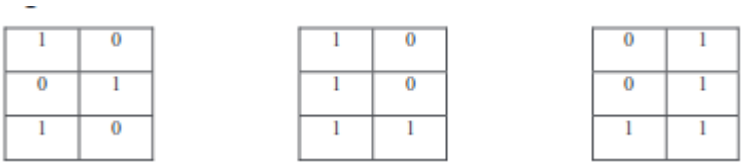
Pomiar różnic pomiędzy szablonami dokonywany jest za pomocą 3 metod: korelacji znormalizowanej, normy ko-sinusowej oraz pomiarze odwzorowania.

Wykorzystanie szablonów opiera się na założeniu, że w obrazie istnieją idealne narożniki.



Rysunek 2.3: Przykłady narożniki.

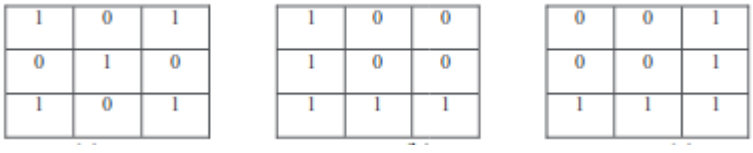
Metoda MBCD polega na wykorzystaniu trzech poniższych szablonów:




Rysunek 2.4: Szablony metody MBCD.

Następnie, każdy z tych szablonów jest obracany o 90 stopni w wyniku czego otrzymujemy 12 szablonów. Punkt jest uznawany za narożnik jeśli pasuje do co najmniej dwóch z pośród wszystkich szablonów.

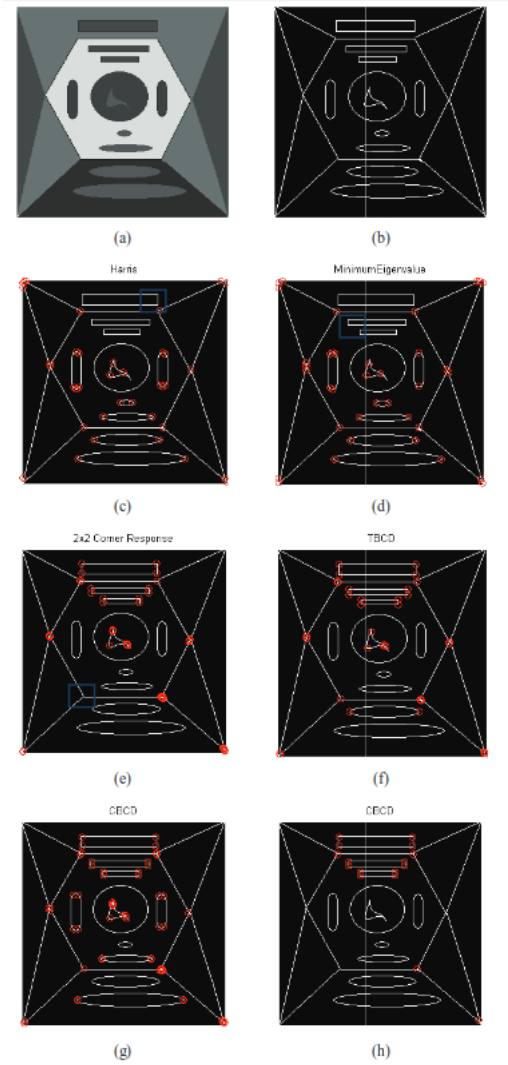
Metoda CBCD posługuje się 3 szablonami:




Rysunek 2.5: Szablony metody CBCD.

Są one obracane o 90 stopni, wy wyniku czego otrzymuje się się 9 szablonów. Piksel jest testowany pod względem podobieństwa z każdy z tych szablonów.

Poniższy obraz prezentuje porównanie algorytmów wykrywania rogów w systemach wizyjnych robotów:




Rysunek 2.6: Porównanie algorytmów.

## 2.3 Ulepszona metoda Harrisa dla wykrywania znaków czcionki chińskiej.

Na podstawie artykułu „Improved Harris Corner Detection for Chinese Characters"(Na Yao, Tiecheng Bai, Xia Jiang, Haifang Lv).

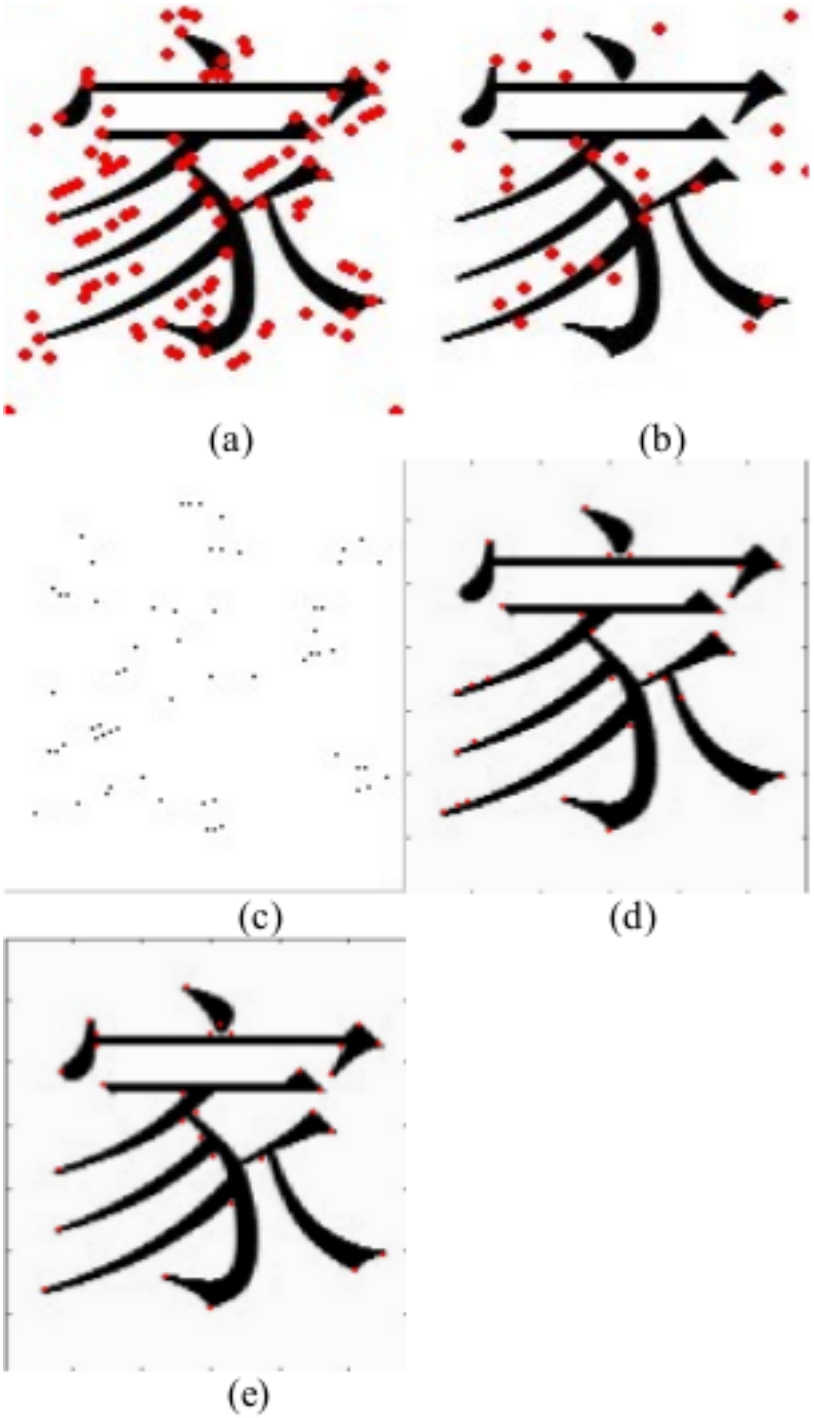
W artykule zaprezentowano metodę badania znaków języka chińskiego. Polegała ona na zastosowaniu algorytmu Harrisa w celu identyfikacji punktów charakterystycznych, a następnie ich weryfikacji za pomocą metody Fast. W artykule znalazło się również porównanie zaproponowanego sposobu z innymi podejściami do badanego problemu.

Wykrywanie i dopasowywanie punktów charakterystycznych jest kluczowym elementem rozpoznawania znaków. Stosunkowo łatwy zadaniem jest odróżnienie znaku od tła i zidentyfikowanie krawędzi. Ważniejszym krokiem jest jednak określenie długości krawędzi i ich punktów końcowych. Wy wyniku wielu eksperymentów udowodniono, że algorytm Harrisa jest najbardziej efektywną metodą wyznaczania punktów charakterystycznych znaków. Niestety, zwraca on również nadmiarowe informacje.

Opracowana metoda składa się z następujących kroków:

1. Obraz RGB jest konwertowany na skalę szarości.
2. Narożniki są wykrywane za pomocą ulepszonego algorytmu Harrisa(zwanego algorytmem Shi-Tomasa).
3. Współrzędne każdego narożnika jest zapisywane.
4. Okrąg o przekątnej 16 pikseli wokół każdego narożnika jest sprawdzany algorytmu Fast.

Poniższy rysunek prezentuje zestawienie różnych metod wyniki metody:

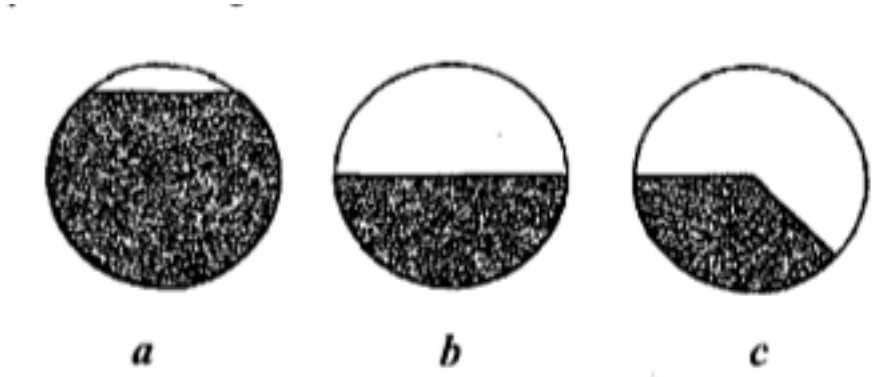



Rysunek 2.7: Wykrywanie punktów charakterystycznych znaków.

## 2.4 Odporny efektywny algorytm wykrywania punktów charakterystycznych.

Na podstawie artykułu „An efficient and robust corner detection algorithm"(Dongxiang Zhou, Yun-hui Liut, Xuanping Cai).

Artykuł prezentuje ulepszoną wersję algorytmu Susan, której działanie porównywane jest z metodą Harrisa. Wykorzystuje ona lokalną binaryzację opartą na kilku progach kilku progach dobieranych na lokalnej jasności punktu. Podobnie jak w oryginalnej metodzie, zliczana jest liczba punktów o zbliżonej jasności. Nowością w proponowanej metodzie jest ograniczenie się do poszukiwania rogów jedynie na krawędziach. Wykryty punkt jest badany za pomocą maski kołowej, zaprezentowanej poniżej:



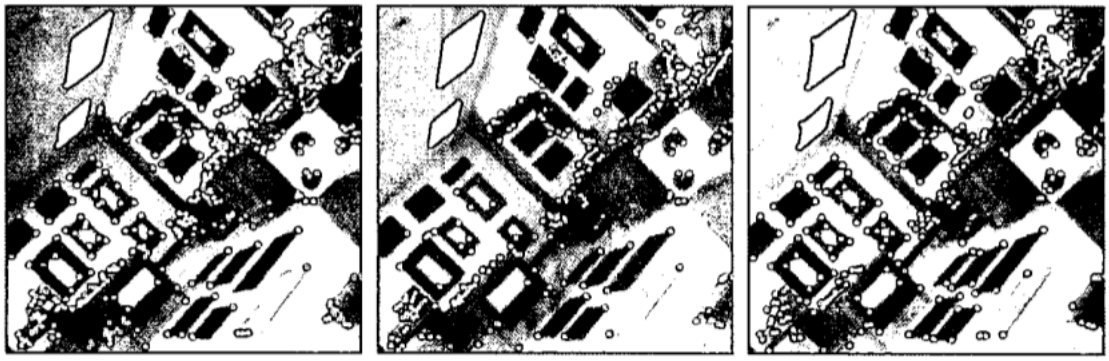

Rysunek 2.8: Maski w algorytmie Susan: a) zwykły punkt, b) krawędź, c) narożnik.

Zaletą stosowanej metody jest odporność na lokalne zakłócenia, gdyż nie wyliczane są żadne gradienty.

Ulepszona metoda SUSAN wprowadza nowy sposób wprowadza nową metodę obliczania progu: opiera się ona na liczbie poziomów szarości i liczbie pikseli mających te poziomy szarości.

Kolejnym usprawnieniem jest podział maski na 2 lub więcej części, w zależności od uzyskanego wcześniej progu.

Rezultat metody prezentuje poniższy obraz:




Rysunek 2.9: Wykrywanie punktów charakterystycznych: a) Harris b)Susan c) ulepszony Susan.

# Rozdział 3

# System detekcji i śledzenia.

## 3.1 Opis algorytmu

## 3.2 Implementacja w C++

# Rozdział 4

# Implementacja C++.

# Rozdział 5

# Podział HW/SW.

# Rozdział 6

# Uruchomienie na Zybo.